# Exercises: State Management

Problems for exercises and homework for the [“Java Web Development Basics” course @ SoftUni](https://softuni.bg/courses/java-web-development-basics). Submit your solutions on the **course page** of the **current instance**.

Let’s see what you’ve learned about Session Management with a simple Application. Use the given skeleton and whatever you can find to help you implement the application.

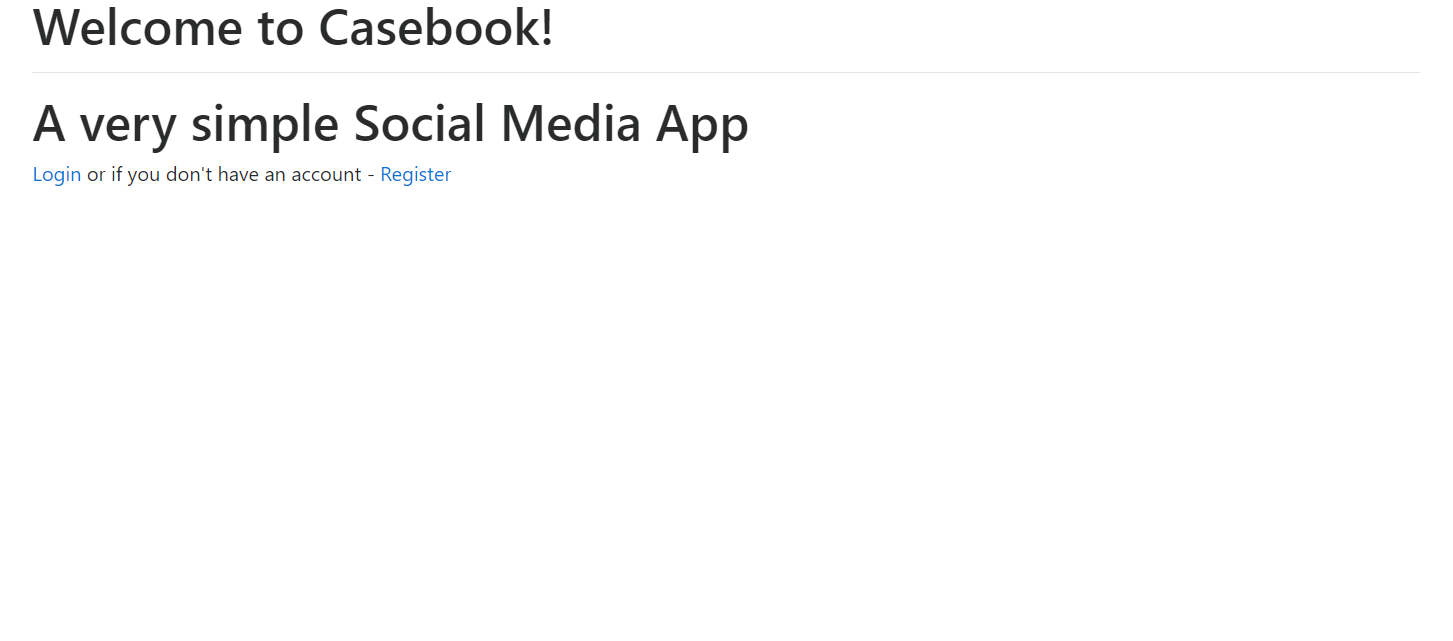
## Skeleton

You will be given the HTTP-Server we’ve been making throughout the lessons, with implemented Session and Cookie logic. The RequestHandler however, will be **empty**, its up to you to **implement** it.

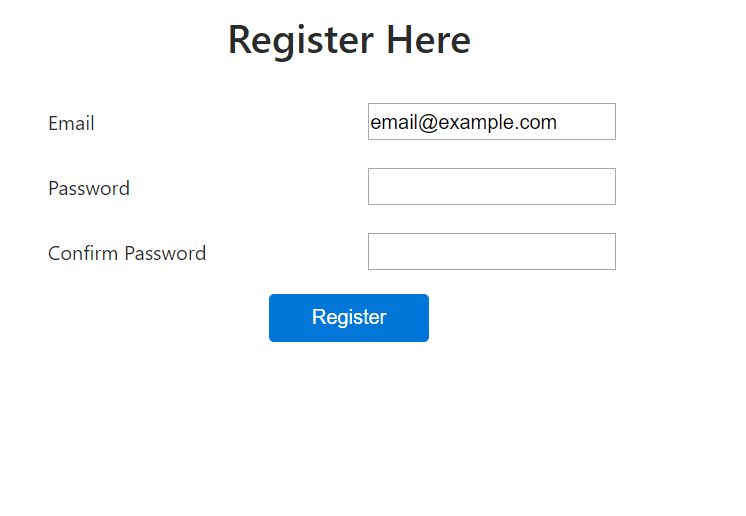
## Pages

You will have several pages – **static** and **dynamic**. Place them in the corresponding folders for **static materials** and for **pages**. Implement **all** **pages** exactly as given below, they should look **EXACTLY** as the screenshots.

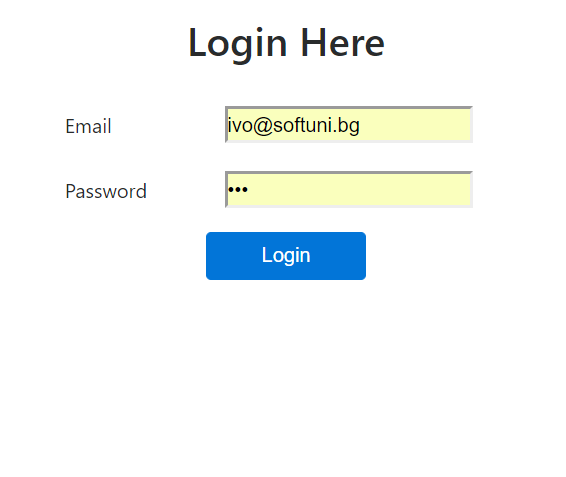
### Index (static) (route=”/”)



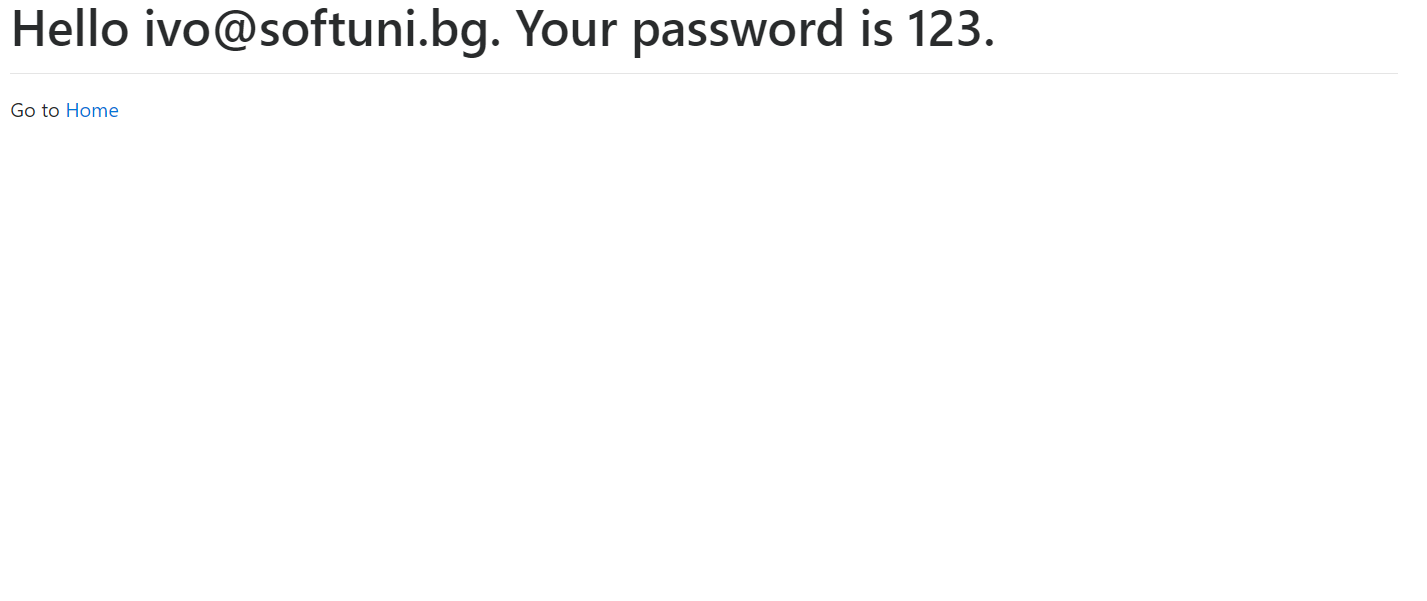
### Register (static) (route=”/html/register.html”)



### Login (static) (route=”/html/login.html”)

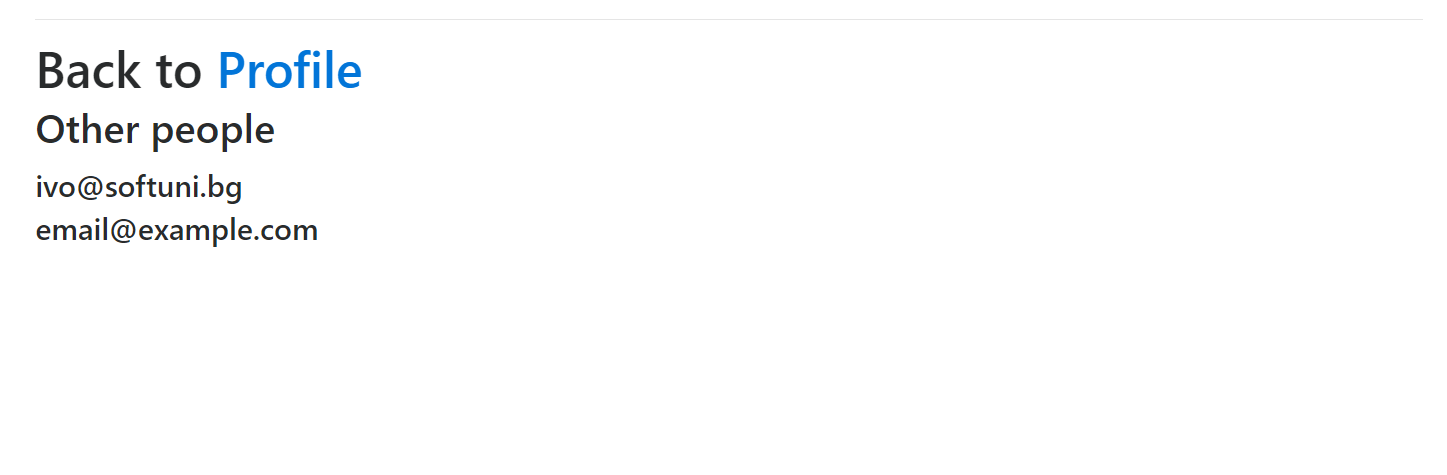


### Profile (dymanic) (route=”/users/profile”)



### Home (dynamic) (route=”/home”)

The home page renders all other registered users’s **names**. (**Other** means **everyone but** the **currently** **logged in user**)



**Static pages** mean, that you just have to load them and send them. **Dynamic pages** actually require you to **render** some **dynamic** **data**.

Use bootstrap classes to implement the **design** of the pages.

## Database

The Casebook application should support Users as its main entities.

* A **user** has an **Id** which is a **String**, preferably UUID.
* A **user** has a **name** which is his **email**, a **String**.
* A **user** has a **password** which is also a **String**.

You should store your entities on a **flat file database** in the db folder of the project.

If you feel the need to use **ANY functionality** for **extracting**, **submitting** or **manipulating data** from the **database**, you should **implement** a **Repository-based** class for that. This is **absolutely required** for the **homework**. Your colleagues are **permitted** to give you a **lesser grade** if you **don’t have** this implementation.

## Business Logic

### Security & Authentication

If **someone** tries to access the /users/profile route or the /home route, **without** being **logged in**, the server should **redirect** him to the Login page.

If a **logged in user** tries to acces the /html/login.html route or the /html/register.html route, the server should **redirect** him to the Profile page.

### Redirection

When a person **registers** **successfully**, he should be **redirected** to the Login page.

When a person **logins successfully**, he should be **redirected** to the Profile page.

### Error Handling

When a request is sent to an invalid URL (different from the ones specified above), you should return a 404 Response, with **Content-Type** – **text/html** and Content – a **heading (1)** with content “Error 404: Page or Resource not found...”;

When a malformed request is sent to your server, you should return a 400 Response, with **Content-Type** – **text/html** and Content – a **heading (1)** with content “Error 400: Malformed Request...”;

## Implement Logout

Implement a logout functionality by your design. Let’s say it is a simple button on the Profile page or Home page.

Upon **logout**, it should **delete** the **user’s cookie** and **terminate** the **session**, and it should **redirect** the client to Index page.